REMARKS

I. Summary of the Examiner's Action

A. Claim Rejections

As set forth in paragraph 2 of the February 9 Office Action, claims 12 – 15 stand rejected under 35 U.S.C. § 101 as being directed to non-statutory subject matter.

As set forth in paragraph 3 of the February 9 Office Action, claims 1 – 22 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over United States Patent No. 5,946,491 to Aizikowitz (hereinafter "the Aizikowitz patent") in view of United States Patent No. 6,487,630 to Bui (hereinafter "the Bui patent").

These rejections are respectfully disagreed with, and are traversed below.

II. Applicant's Response

A. Rejection of Claims 12 – 15 under 35 U.S.C. § 101

Applicants respectfully disagree with Examiner's rejection of claims 12 – 15. Notably, claim 12 recites "a computer usable medium having computer readable program code embodied therein for managing data in a register ..." Since claim 12 refers to a computer usable medium that embodies a computer readable program, it is directed to statutory subject matter. Applicants respectfully refer the Examiner to the ordinary definition of "embodies" which means to make incarnate or concrete. *See*, for example, the Merriam-

Webster Online Dictionary.

B. Rejection of Claims 1 - 22 under 35 U.S.C. § 103(a)

Claim 1 recites:

> 1. A method for managing data in a register, comprising:
> introducing a name level instruction for at least one of a named architected register in a processor;
> allowing a programmer to change the current name level of a register name via said name level of instruction;
> creating a new register with an internal name and a new name level, and
> providing a plurality of additional available computer registers.

It is not seen where the art relied upon by the Examiner, whether taken singly or in combination, has anything to do with the subject matter of claim 1. In fact, whereas the present invention concerns methods and apparatus allowing programmers to benefit from operations similar to register renaming, which reduce the need for register spilling, the Aizikowitz is *per se* concerned with methods for improving spilling operations. In other words, Aizikowitz is not concerned with methods that increase the number of registers available to a user, and thereby avoid or at least reduce the need for register spilling.

Aizikowitz's concern with spilling operations is evident both from its title, and its summary section at Column 2, lines 38 - 51, reproduced here (emphasis added):

"According to the present invention, a register allocation method and apparatus efficiently allocates the processor registers in a computer system to program variables in a computer program in a manner that minimizes spill code by accounting for the register pressure when making spill decisions and favoring regions of high register pressure for spilling, thereby avoiding the insertion of spill code in low register pressure regions. By creating spill code in high pressure regions and avoiding spill code in low pressure regions in accordance with the present invention, less spill code is introduced, resulting in more efficient allocation of registers and enhanced run-time performance of the computer program."

In contrast, Applicants' invention concerns methods and apparatus that avoid the need for spill code by making more registers available to a programmer, as described at page 13, line 1 – page 15, line 3 of the Application reproduced here (emphasis added):

"The invention disclosed herein presents a new method of reducing the overhead of register management. One embodiment of the invention introduces a concept of name level for each of the named architected registers in the processor. When the machine begins execution each of the register names has the level of '0' associated with it. The invention introduces a new instruction called 'nameLev', which allows the programmer to change the 'current' name level of a register name. Referring to FIG. 4, which shows an example of when a programmer runs out of usable registers after the use of R1. In this case, the programmer simply inserts the following nameLev instruction:

nameLev R1, 1

which associates the new name level of '1' with the architected register name R1. From that point on until the programmer changes the name level of R1

back to "0", a new register named R1 will be available for use. The program follows and uses the new R1, as needed. When the new R1 is no longer required or the 'old' register R1 must be used, the programmer inserts the instruction:

nameLev R1, 0

at that point in the program. From that point on, when a program instruction uses the name R1, it refers to R1 at name level 0. FIG. 5 shows what happens inside the processor when the example sequence is presented for execution. When execution of this piece of code begins, the LOAD R1 instruction creates an internal name for R1 as R80. At the first use of R1, internal register R80 is used to supply the desired value in R1. An instruction then executes that requires an architected register name, but all register names (in this 2-register machine example) are taken. The programmer then inserts the namelev R1, 1 instruction. The nameLev instance informs the machine that the programmer intends to use a new internal register for R1. The machine uses its internal name management method (in an out-of-order with renaming processor, a rename map table), to remember the change. The next time an instruction uses the name R1 as one of its source registers, the newly remembered internal register is used to supply the source operand for that instruction. On the other hand, if and when an instruction writes its results into R1 (not shown in this example), the results are written to a new physical register assigned by the hardware in the normal course of execution.

To summarize, an extra named register was made available to the programmer on demand and when the use of the extra register was no longer needed, it was taken out of use. The method used the exact same encoding of instructions as in FIG. 3, with the only additional overhead being imposed was that of the two extra instructions. The process of mapping the new name levels to internal resources was completely managed by the processor

> implementation and is completely hidden from the programmer. Thus, the original goal of an architectural specification, which allows the programmer complete independence from the machine implementation while guaranteeing the correctness of program execution, is achieved."

The provision of what is effectively an extra architected register thus avoids the need for unnecessary spill code when there exists more physical registers in the processor than architected registers.

The advantages of Applicants' invention are summarized at page 18, line 3 – page 19, line 5, reproduced here (emphasis added):

> "Some of the advantages of the present invention are as follows. This invention makes more architected registers available to the programmer, via overloading or extending the register namespace dynamically under program control. At least as many as the number of physical registers in the processor are available for direct use by the programmer at low access latencies. Any program that uses more registers than that could experience performance penalties, but will still execute as expected by the programmer. In other words, the programmer now has the flexibility to trade-off performance for assured accuracy.
>
> * * *
>
> Further, the present invention reduces the power consumption in the processor by reducing the unnecessary traffic to the storage hierarchy, which is necessitated by the spill and re-fill instructions in processors without the present invention.
>
> Further, improved performance of an application on a processor that

> uses the present invention is achieved by reducing unnecessary spill and re-fill instructions, thereby reducing the burden on the execution pipeline of the processor, the data memory hierarchy of the processor, and the cache coherence traffic in the system."

Returning to the outstanding rejection, the Examiner has apparently confused the concepts of name level instructions and register spilling as is apparent from page 3, lines 1 – 3 of the February 9 Office Action (emphasis added):

> "a) Introducing a name level instruction for at least one of a named architected register in a processor (e.g., see col. 5, lines 1 – 48)[spill code instructions are introduced into instruction stream for at least one physical architected register];
>
> b) Allowing a programmer to change the current name level of a register name via said name level instruction (e.g., see col. 5, lines 1 – 28 and col. 6, lines 8 – 67)[the level of the register name is changed from the level of the level of the physical register in maximum use to the level of the backing store, because the operation is performed in a programmed computer it is allowed to be programmed in the system by the programmer that writes the program]; ..."

Notably, name level instructions as claimed by Applicants are *not* spill code. Spill code writes register contents to memory when there is a shortfall of available registers. Applicants' name level instructions, as explained previously, do not write register contents to memory when there is a register shortfall; rather, the name level instructions make additional registers available to a programmer. Further, the portions of the Aizikowitz patent

referenced by the Examiner refer to methods for adding spill code, and not to adding a name level instruction as in the case of Applicants' invention.

This made apparent by the section of Aizikowitz appearing at Column 5, lines 1 – 28 (emphasis added):

> "Physical processor registers are then allocated to the live ranges using a graph coloring technique that is well known in the art. If all the live ranges of symbolic registers may be allocated to physical processor registers, the optimizing compiler produces a machine code instruction stream without spill code. If one or more of the symbolic registers cannot be allocated to a processor register, the live range must be 'spilled', meaning that the live range is allocated to a memory location rather than to a register, and therefore must be loaded into a register from memory before use, and must be written back to memory after being changed. Loads and stores to memory take considerably longer than operations to registers, and minimizing the number of loads and stores to memory is thus a primary goal of an optimizing compiler in order to minimize the execution time of the machine code instruction stream. If the live range is spilled, spill code (i.e., memory loads and stores) must be added to the intermediate language instruction stream to accomplish the required accesses to memory. The loading and storing of spilled live ranges adds overhead to the machine code instruction stream, slowing its execution time, and slows compilation time due to the extra processing to make decisions concerning the insertion of spill code. Therefore, an optimizing compiler typically has a goal of efficiently allocating processor registers to the highest number of symbolic registers possible while minimizing spill code in order to minimize both the compile-time and run-time overhead associated with spill code."

As stated previously, the portions relied upon by the Examiner refer to spill code and methods for adding spill code, and not to a method for making more registers available to programmers as in the case of Applicants' invention. In addition, the just-reproduced portion of the Aizikowitz patent refers to operations performed by an optimizing compiler, and not by a programmer, so it is not seen how it relates to the subject matter of claim 1, which refers to operations that allow a programmer to change the name level of a register.

Bui is not seen to overcome the deficiencies of the Aizikowitz patent as set forth in the foregoing remarks.

For the foregoing reasons Applicants respectfully submit that claim 1 is patentable over the art of record, whether taken singly or in combination. Accordingly, Applicants respectfully request that the rejection of claim 1 be withdrawn. Applicants respectfully submit that independent claims 12 and 16 are patentable both for reasons similar to claim 1 and for reasons attributable to their independently-recited features. Applicants therefore respectfully request that the rejection of independent claims 12 and 16 be withdrawn as well. Further, applicants respectfully submit that dependent claims 2 – 11; 13 – 15; and 17 – 22 are allowable both as depending from allowable base claims and for reasons attributable to their independently-recited features. Therefore, Applicants respectfully request that the rejection of dependent claims 2 – 11; 13 – 15; and 17 – 22 be withdrawn as well.

III. Conclusion

The Applicants submit that in light of the foregoing remarks the application is now in condition for allowance. Applicants therefore respectfully request that the outstanding rejections be withdrawn and that the case be passed to issuance.

Respectfully submitted,

May 8, 2006
Date

David M. O'Neill (35,304)
David M. O'Neill (Reg. No. 35,304)

Customer No.: 29683
HARRINGTON & SMITH, LLP
4 Research Drive
Shelton, CT 06484-6212
Telephone: (203)925-9400
Facsimile: (203)944-0245
email: DOneill@hspatent.com

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

5/8/2006
Date

Elaine F. Main
Name of Person Making Deposit